1991

A Framework for 1-D Compaction with Forbidden Region Avoidance

Susanne E. Hambrusch
Purdue University, seh@cs.purdue.edu

Hung-Yi Tu

Report Number:
91-014

A FRAMEWORK FOR 1-D COMPACTION WITH FORBIDDEN REGION AVOIDANCE

Susanne Hambrusch
Hung-Yi Tu

CSD-TR-91-014
February 1991

A Framework for 1-D Compaction with Forbidden Region Avoidance

Susanne Hambrusch *
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Hung-Yi Tu†
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

March 18, 1991

Abstract

In this paper we consider the 1-dimensional compaction problem when the layout area contains forbidden regions and the layout components are allowed to move across these regions. Assume we are given a feasible layout containing k forbidden regions and n layout components, where the i-th layout component is a rectilinear polygon consisting of v_i vertical edges, $v = \sum_{i=1}^{n} v_i$. We present an algorithm that determines the positions of the layout components resulting in minimum area in $O(\sigma \log \sigma + \sigma n \log n)$ time with an $O((v+k)\log k + (v+\sigma)\log v)$ preprocessing time. The quantity σ measures the interaction between the layout components and the forbidden regions, $\sigma \leq vk$. We also describe variants of this algorithms that make the running time more problem-dependent and consider forbidden regions of special structure. Our algorithms make use of an elegant characterization of a layout of minimum area.

*Research supported in part by ONR under contracts N00014-84-K-0502 and N00014-86-K-0689, and by NSF under Grant MIP-87-15652.

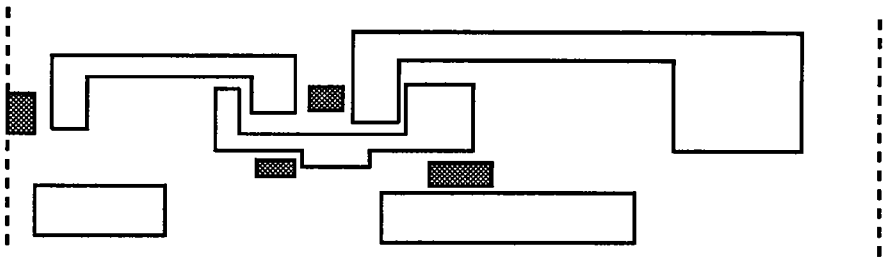
†Research supported in part by NSF under Grant MIP-87-15652 and ONR under contract N00014-84-K-0502.

1 Introduction

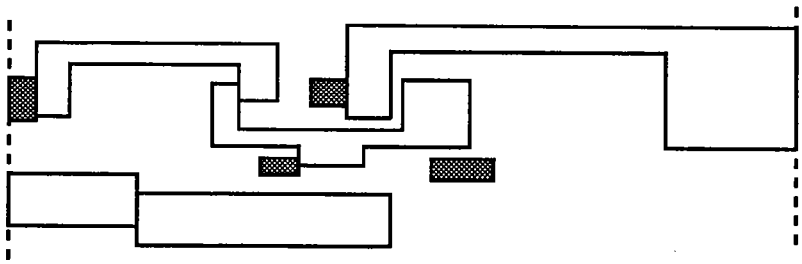
A one-dimensional (1-D) compactor takes as an input a VLSI layout and generates a layout of smaller area by sliding the layout components in one direction [2, 3, 7, 9]. W.l.o.g., let it be the horizontal direction. In this paper we consider the compaction process when the layout area contains forbidden regions. The forbidden regions can represent, for example, pre-positioned layout components or holes in the layout area. Being able to handle forbidden regions is a natural generalization and the presence of forbidden regions in routing problems has been studied [5]. The positions of the forbidden regions cannot be altered during the compaction process, but layout components are allowed to "slide over" the forbidden regions. We develop a general method for performing 1-D compaction with forbidden region avoidance.

Given are n rectilinear polygons, $P_1, P_2, \ldots, P_n$, and k forbidden regions, $B_1, B_2, \ldots, B_k$, with the edges of the polygons and forbidden regions parallel to the coordinate axes. Polygon P_i consists of v_i vertical edges, $v_i \geq 2$, with $v = \sum_{i=1}^{n} v_i$. W.l.o.g., we assume that the forbidden regions are rectangles. Straightforward modifications to the algorithms can handle forbidden regions of rectilinear polygonal shape. A *configuration* C of the layout assigns to the leftmost vertical edge of every polygon an x-position. A configuration is called *feasible* if it keeps the relative order of the polygons in the horizontal direction and no two polygons and no polygon and forbidden region overlap. A feasible configuration of minimum area is called a *minimum configuration*. The objective of the *forbidden region problem* is to find, for a given feasible configuration, a minimum configuration. Figure 1(a) shows a feasible configuration and Figure 1(c) shows a minimum configuration.

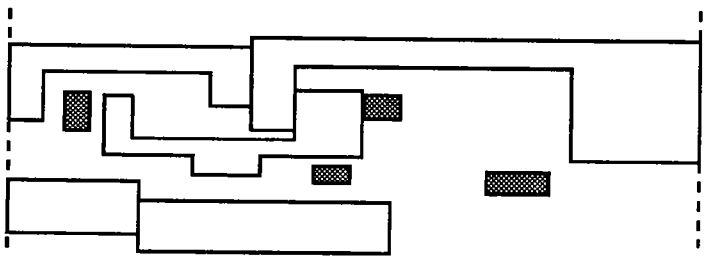
Before stating our results we define a quantity σ that captures the interaction between polygons and forbidden regions and which will enter our running times.



(a) A feasible configuration



(b) The left-compressed version of (a)



(c) A minimum configuration

Figure 1: A forbidden region problem

Let e_j be the left vertical edge of forbidden region B_j and let $e_{i,q}$ be a vertical edge of polygon B_i, $1 \leq q \leq v_i$. We say forbidden region B_j and edge $e_{i,q}$ are *related* if there exists a horizontal line that intersects both e_j and $e_{i,q}$. Let $s_{i,q}$ be the number of forbidden regions related to edge $e_{i,q}$, $s_{i,q} \leq k$. Let s_i be the sum over all $s_{i,q}$'s, where $e_{i,q}$ is a vertical edge of polygon P_i (summing over all q's). To simplify boundary cases we assume $s_i \geq 1$. Then, $\sigma = \sum_{i=1}^{n} s_i$, $n \leq \sigma \leq vk$.

For convenience we introduce a fictitious polygon P_0 of rectangular shape having width 0 and height h, where h is the height of the layout. (Since compaction is done in the horizontal direction, h is determined by the forbidden regions and the polygons, and is not altered during compaction.) Assume we know the position of polygon P_0 in the layout area. For arbitrary polygons we show how to determine a minimum configuration for this particular position of P_0 in $O(\sigma n \log n)$ time with $O(\sigma \log \sigma + (v+k) \log k + (v+\sigma) \log v)$ preprocessing time. The preprocessing time includes setting up data structures used throughout the algorithm. We also present a faster, more problem-dependent version of this algorithm. When every polygon is a horizontally convex polygon (i.e., no horizontal line intersects the polygon more than once), our algorithm determines a minimum configuration in $O(\sigma)$ time, using the same preprocessing time.

Determining the position of P_0 in a minimum configuration is the heart of our forbidden region algorithms. We determine P_0's position by (i) characterizing at most σ feasible configurations, each of which has a fixed position of P_0 associated with it, (ii) showing that a minimum configuration is among these σ configurations, and (iii) generating these configurations in an order that allows us to update changes in the width of each configuration efficiently. Once the position of P_0 resulting in a minimum configuration is known, we use the above described algorithm to determine the positions of the n polygons. For arbitrary polygons we present an algorithm for the forbidden region problem that runs in $O(\sigma \log \sigma + \sigma n \log n)$ time with an additional $O((v+k) \log k + (v+\sigma) \log v)$ time

for preprocessing. Hence, determining the width of σ configurations is, in the worst-case, no more expensive than determining the width of one configuration in which the position of P_0 fixed. For convex polygons the forbidden region problem can be solved in $O(\sigma n)$ time. We describe a number of variants of this algorithm that make the running time more problem-dependent. We also consider the compaction problem when every forbidden region has height h. We call this problem the *k-partition problem.* The k forbidden regions now model positions in the layout area where the layout can be "cut" by a straight line.

This paper is structured as follows. In Section 2 we state definitions and describe some preprocessing steps. In Section 3 we present the algorithm that determines a minimum configuration when the position of P_0 is fixed. Section 4 presents the algorithms for the forbidden region problem. In Section 5 we consider the k-partition problem. Section 6 concludes the paper.

2 Definitions and Preliminaries

In this section we give some definitions and describe data structures used throughout. In addition to polygon P_0 we use another fictitious polygon, P_{n+1}, also a rectangle of height h and width zero. In all feasible configuration we require that P_0 and P_{n+1} are positioned to the left and to the right of the other polygons and forbidden regions, respectively. Let $x(B_j)$ be the position (i.e., the x-coordinate of the vertical left side) of forbidden region B_j. In any configuration $\mathcal{C}$, let $x_{\mathcal{C}}(P_i)$ be the position of the leftmost vertical edge of polygon P_i. Also, for any edge $e_{i,q}$ of P_i, let $x_{\mathcal{C}}(e_{i,q})$ be the position of edge $e_{i,q}$ in configuration $\mathcal{C}$. The width of configuration $\mathcal{C}$ is defined by the distance between P_0 and P_{n+1}; i.e., $x_{\mathcal{C}}(P_{n+1}) - x_{\mathcal{C}}(P_0)$.

A feasible configuration $\mathcal{C}$ is called *left-compressed* if for any other feasible configuration $\mathcal{C}'$ in which $x_{\mathcal{C}'}(P_0) = x_{\mathcal{C}}(P_0)$ we have $x_{\mathcal{C}}(P_i) \leq x_{\mathcal{C}'}(P_i)$ for $1 \leq i \leq n+1$. Intuitively, in a left-compressed configuration all polygons P_i, $1 \leq$

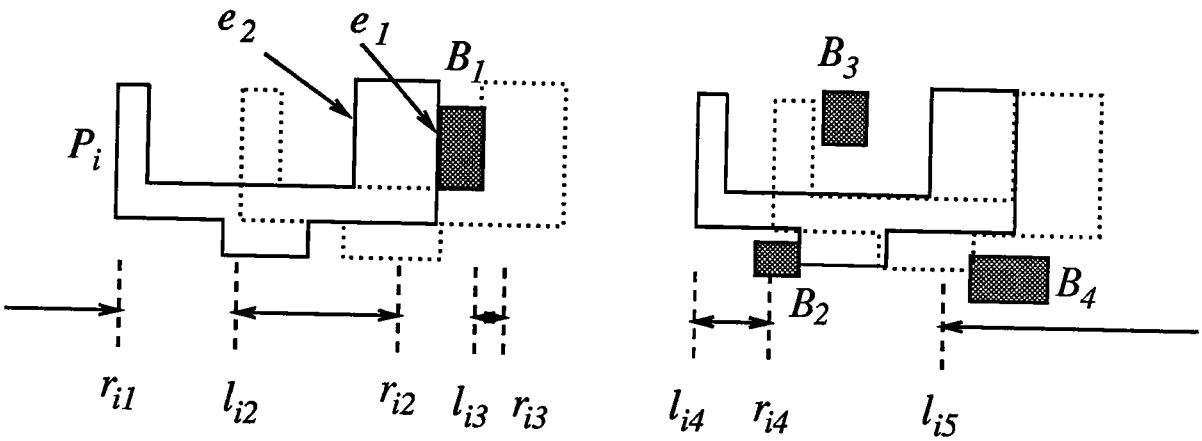


Figure 2: Determining the slot list for P_i

$i \leq n+1$, are positioned as far to the left as possible. It is easy to see that performing a left-compression on a given configuration does not increase its width. Figure 1(b) shows the left-compressed version of the configuration shown in Figure 1(a).

We say position $x_C(P_i)$ is *legal* for P_i if polygon P_i, when assigned to this position, does not overlap with any forbidden region. In all our algorithms polygon P_i needs to know which positions are legal for it. Let $\mathcal{S}_i$ be a list containing all legal positions for P_i, sorted by increasing x-coordinates. These positions can be represented as mutually disjoint intervals. List $\mathcal{S}_i$ contains at least one and at most s_i intervals, where s_i is the quantity defined in Section 1. The t-th interval is represented by its left endpoint $l_{i,t}$ and its right endpoint $r_{i,t}$. We refer to $\mathcal{S}_i$ as the *slot list* of polygon P_i and throughout $\mathcal{S}_i$ is represented as a list of intervals. In order to keep P_0 and P_{n+1} to the left and to right of all polygons and forbidden regions, respectively, $\mathcal{S}_0$ consists of the interval $(-\infty, x(B_1)]$ and $\mathcal{S}_{n+1}$ of $[x(e_{right}), +\infty)$, where $x(e_{right})$ is the position of the rightmost vertical edge of the forbidden regions.

Figure 2 shows the endpoints of the intervals in the slot list for a polygon P_i. Polygon P_i has $v_i = 6$, $s_i = 12$ (every one of its 6 vertical edges is related to 2 forbidden regions), and $\mathcal{S}_i$ consists of 5 intervals, as indicated. For the 4-th interval, P_i is shown with solid edges positioned with $x(P_i) = l_{i,4}$ and it is shown with dotted edges positioned with $x(P_i) = r_{i,4}$. Any position beween $l_{i,4}$ and $r_{i,4}$ is legal for P_i.

We next describe how to determine the slot list $\mathcal{S}_i$ efficiently. A first step determines for every vertical edge of polygon P_i the forbidden regions related to this vertical edge. Let $e_{i,q}$ be a vertical edge of P_i and let $y_{q,1}$ and $y_{q,2}$ be the y-coordinates of the two endpoints of edge $e_{i,q}$. Let $y_{j,1}$ and $y_{j,2}$ be the y-coordinates of the endpoints of the left vertical edge of forbidden region B_j. The problem of determining the forbidden regions related to $e_{i,q}$ can be stated as follows. Given a set V of line segments with $V = \{[y_{j,1}, y_{j,2}] | 1 \leq j \leq k\}$ and a query line segment $e_{i,q} = [y_{q,1}, y_{q,2}]$, determine the segments in V that have a point in common with $e_{i,q}$. Using a binary search tree, the forbidden regions related to edge $e_{i,q}$ can be determined in $O(s_{i,q} + \log k)$ time, where $s_{i,q}$ is the number of the forbidden regions related to vertical edge $e_{i,q}$, [10]. The binary search tree is static and is built in $O(k \log k)$ time once for all v vertical edges of the n polygons.

Every forbidden region B_j related to a vertical edge $e_{i,q}$ of polygon P_i induces two positions for polygon P_i. The *right-value* $xr_{j,q}(P_i)$ is the position of polygon P_i in which $x(e_{i,q}) = x(B_j)$ (i.e., the position of edge $e_{i,q}$ coincides with the left vertical edge of forbidden region B_j). If positioning P_i at $xr_{j,q}(P_i)$ does not cause an overlap between P_i and B_j, then $xr_{j,q}(P_i)$ is a legal right-value. Legal left-values are defined in a similar way. The *left-value* $xl_{j,q}(P_i)$ is the position of polygon P_i in which $x(e_{i,q}) = x(B_j) + w(B_j)$, where $w(B_j)$ is the width of forbidden region B_j. If positioning P_i at $xl_{j,q}(P_i)$ does not cause an overlap between P_i and B_j, then $xl_{j,q}(P_i)$ is a legal left-value. For example, edge e_1 of P_i and forbidden region B_1 shown in Figure 2 induce a legal right-value corresponding to the position labeled $r_{i,1}$. In the position corresponding to the left-value induced by e_1 and B_1 polygon P_i and B_1 overlap. Hence, e_1 and B_1 do not induce a legal left-value. Edge e_2 and B_1 do induce a legal left-value corresponding to $l_{i,2}$. Let $\mathcal{L}_i$ be the list containing the legal right- and left-values of all vertical edges of P_i, sorted by non-decreasing x-positions.

Whether a particular right- or left-value induced by P_i and B_j is legal (i.e., P_i and B_j do not overlap) can be determined in $O(\log v_i)$ time, where v_i is the number of vertical edges of P_i. The overlap check uses a static data structure that is built in $O(v_i \log v_i)$ time and this data structure is used by all vertical edges of polygon P_i. From the above discussion it follows that list $\mathcal{L}_i$ is generated in $O(v_i \log k + s_i \log v_i + s_i \log s_i)$ time.

The slot list $\mathcal{S}_i$ contains all legal positions for P_i represented by intervals and it is generated by scanning list $\mathcal{L}_i$ which contains all legal right- and left-values. The first interval of $\mathcal{S}_i$ is determined easily: we set $l_{i,1} = -\infty$ and $r_{i,1}$ to the x-position associated with the first element in list $\mathcal{L}_i$. This first x-position corresponds to a legal right-value. In Figure 2, $r_{i,1}$ corresponds to the legal right-value induced by e_1 and B_1. Assume we have determined the $(t-1)$-st interval. The right endpoint $r_{i,t-1}$ of this interval always correspond to a legal right-value in list $\mathcal{L}_i$. Let $xr_{j,q}(P_i)$ be this right-value. The left endpoint of t-th interval is determined as follows. Let $xl_{j,p}(P_i)$ be the smallest legal left-value induced by B_j and some edge $e_{i,p}$ of P_i so that $xl_{j,p}(P_i) \geq xr_{j,q}(P_i)$. The difference between $xl_{j,p}(P_i)$ and $xr_{j,q}(P_i)$ represents the minimum distance that P_i needs to move to the right until it is again positioned so that P_i and forbidden region B_j do not overlap. In this position forbidden region B_j is immediately to the left of edge $e_{i,p}$ of P_i. However, this position may not be a legal position, since P_i may now overlap with another forbidden region. If there are no right-values between $xr_{j,q}(P_i)$ and $xl_{j,p}(P_i)$ in list $\mathcal{L}_i$, then no other forbidden region overlaps with P_i when P_i is positioned at $xl_{j,p}(P_i)$. In this case we set $l_{i,t} = xl_{j,p}(P_i)$. With respect to Figure 2, when $t = 1$, the left-value induced by edge e_2 and forbidden region B_1 gives the minimum distance P_i has to move to the right. Since there are no other right values between $r_{i,1}$ and this value, we found the left endpoint of the second interval.

Assume now that there exists a right-value $xr_{j',q'}(P_i)$'s with $xr_{j,q}(P_i) \leq$

$xr_{j',q'}(P_i) < xl_{j,p}(P_i)$. Let $xl_{j',p'}(P_i)$ be the smallest legal left-value larger than $xr_{j',q'}(P_i)$. In other words, position $xl_{j',p'}(P_i)$ is the closest position to the right of $xr_{j',q'}(P_i)$ in which P_i and forbidden region $B_{j'}$ do not overlap. If there exists more than one such right-value in this range, we choose j' such that $xl_{j',p'}(P_i)$ is maximized. If $xl_{j',p'}(P_i)$ is smaller than $xl_{j,p}(P_i)$, then every illegal position between $xr_{j',q'}(P_i)$ and $xl_{j',p'}(P_i)$ is contained in the set of illegal positions between $xr_{j,q}(P_i)$ and $xl_{j,p}(P_i)$. Hence, we set $l_{i,t} = xl_{j,p}(P_i)$. If we have $xl_{j',p'}(P_i) > xl_{j,p}(P_i)$, the left-value $xl_{j,p}(P_i)$ cannot be the left endpoint of the next interval. In this case we continue our search for a left endpoint $l_{i,t}$ with index j'.

Once $l_{i,t}$ has been determined, the right endpoint $r_{i,t}$ is determined by locating in list $\mathcal{L}_i$ the smallest right-value larger than $l_{i,t}$. The process of creating intervals continues until all entries in the related list have been handled. Given list $\mathcal{L}_i$, the slot list $\mathcal{S}_i$ is generated in $O(s_i)$ time. In summary, the slot lists for the n polygons are created in $O(\sigma \log \sigma + (v+k)\log k + (v+\sigma)\log v_{max})$ time, where $v_{max} = \max_{1 \le i \le n} v_i$.

We conclude this section by defining two visibility graphs, one on the polygons, and one on the strongly connected components of the polygon visibility graph. A vertical edge $e_{i,r}$ is *visible* from a vertical edge $e_{j,q}$ iff $i \neq j$, $e_{i,r}$ is to the right of $e_{j,q}$, and one can draw a horizontal line segment connecting them without intersecting any other edge of a polygon. Let $G_p = (V_p, A_p)$ be the visibility graph induced by the polygons, which we call the *polygon graph*. Graph G_p consists of $n+2$ vertices and vertex u_i corresponds polygon P_i. The arc (u_i, u_j) is in G_p iff there exists a vertical edge $e_{i,q}$ of P_i and a vertical edge $e_{j,r}$ of P_j such that $e_{j,r}$ is visible from $e_{i,q}$. Graph G_p is planar, but it may have cycles. Vertex u_0 is the source and u_{n+1} is the sink of G_p. Using standard techniques (e.g., a sweep-line approach together with a balanced tree structure to support queries on intervals), graph G_p can be generated in $O(v \log v)$ time. Given a

configuration $\mathcal{C}$, we associate with every arc of G_p a weight. The weight of arc (u_i, u_j) is the distance between P_i and P_j (i.e., the smallest distance between an edge $e_{i,q}$ of P_i and $e_{j,r}$ of P_j with $e_{j,r}$ visible from $e_{i,q}$).

Let $C_1, C_2, \ldots, C_\alpha$ be the strongly connected components of G_p. The *component graph* G_c consists of α vertices, each vertex representing a strongly connected component of G_p. There exists an arc from u_i to u_j in G_c iff G_p contains at least one arc from a vertex in component C_i to a vertex in component C_j. Obviously, G_c is acyclic and planar. G_c can be generated from G_c in $O(n)$ time.

3 Compacting when P_0 is fixed

Assume we are given a feasible configuration $\mathcal{C}$ in which polygon P_0 is positioned to the left of the k forbidden regions and all other polygons are positioned to the right of the forbidden regions. In this section we describe how to generate $\mathcal{C}_{left}$, the left-compressed version of configuration $\mathcal{C}$. Assume we have determined the polygon graph G_p and computed the weights of its arcs with respect to configuration $\mathcal{C}$. Intuitively, the weight of the arc (u_i, u_j) in G_p represents the maximum amount polygon P_j can move to the left in configuration $\mathcal{C}$ without overlapping with polygon P_i when P_i is kept fixed and the forbidden regions are not taken into account. We determine configuration $\mathcal{C}_{left}$ by computing for every polygon P_i a quantity $ml(i)$ which represents the amount polygon P_i moves to the left from its position in configuration $\mathcal{C}$ to its position in configuration $\mathcal{C}_{left}$.

When no forbidden regions are present, it is easy to see that $ml(i)$ is the length of the shortest path from vertex u_0 to u_i in G_p. Hence, in this case, ignoring necessary preprocessing steps, configuration $\mathcal{C}_{left}$ can be computed in $O(n \log n)$ time [1]. In order to determine the ml-entries when forbidden regions are present, we perform a Dijkstra-like shortest path computation on G_p. The existence of forbidden regions will cause already computed shortest paths entries to get reduced, similar to a shortest path computation in a graph containing

negative edge weights, but no negative cycles.

Initially we set $ml(0) = 0$, $ml(i) = +\infty$, and we set the weights $w_{i,j}$ in G_p with respect to configuration $\mathcal{C}$. Assume, furthermore, that for every polygon P_i we have determined its slot list $\mathcal{S}_i$. Our algorithm maintains a set S of vertices whose corresponding polygons have temporary left-compressed positions assigned to them. Initially S is empty. The algorithm repeatedly selects a vertex u_i with the minimum ml-entry from $V_p - S$, inserts u_i into S, and performs a relaxation step and a legality test on all arcs leaving u_i. The relaxation step is as in the shortest paths algorithm: for every arc (u_i, u_j) set $ml(j) = \min\{ml(j), ml(i) + w_{i,j}\}$. The legality test checks whether the position $x_{\mathcal{C}}(P_j) - ml(j)$ is legal. If it is not, we determine the closest legal position for P_j to the right of its current position. This position corresponds to a left endpoint of an interval in list $\mathcal{S}_i$. The new, legal position determined causes $ml(j)$ to be decreased. Whenever $ml(j)$ is decreased and vertex u_j is in set S, we delete u_j from S and add it to set Q. The algorithm terminates when for every polygon P_j the position $x_{\mathcal{C}}(P_j) - ml(j)$ is legal.

Figure 3 gives a detailed description of algorithm FIXP0. It maintains a priority queue Q containing vertices organized by their ml-entries and ExtractMin(Q) selects the smallest entry from Q. To prove the correctness of algorithm FIXP0, we show that the configuration generated is feasible and left-compressed.

Lemma 3.1 *Let $\mathcal{C}'$ be the configuration obtained by setting $x_{\mathcal{C}'}(P_i) = x_{\mathcal{C}}(P_i) - ml(i)$. Then, $\mathcal{C}' = \mathcal{C}_{left}$.*

Proof: Recall that a feasible configuration satisfies three conditions. First, there are no intersections between the polygons and forbidden regions; second, the relative order of the polygons did not change, and third, there are no intersections among the polygons. Since $ml(i)$ is initialized with $+\infty$, a legality test is

Algorithm **FIXP0:**
Input: Graph G_p with weights set with respect to configuration $\mathcal{C}$
Output: Vector ml

1. Initialize ml-entries: $ml(0) = 0$, $ml(i) = +\infty$ for $1 \leq i \leq n+1$;
2. $S \leftarrow \emptyset$;
3. $Q \leftarrow V_p$;
4. **while** $Q \neq \emptyset$ **do**
   begin
5. $u_i \leftarrow \text{ExtractMin}(Q)$;
6. $S \leftarrow S \cup \{u_i\}$;
7. **for** each vertex u_j such that $(u_i, u_j) \in A_p$ **do**
   begin
8. **if** $ml(j) > ml(i) + w_{i,j}$ **then**
   begin
9. $ml(j) \leftarrow ml(i) + w_{i,j}$;
10. **if** $x_{\mathcal{C}}(P_j) - ml(j)$ is not legal **then**
   begin
11. determine the smallest q such that $x_{\mathcal{C}}(P_j) - ml(j) < l_{j,q}$;
12. $ml(j) \leftarrow x_{\mathcal{C}}(P_j) - l_{j,q}$;
   end (* then *)
13. **if** $u_j \in S$ **then** $S \leftarrow S - \{u_j\}$; $Q \leftarrow Q \cup \{u_j\}$;
   end(* then *)
   end (* for *)
   end (* while *)

Figure 3: Algorithm FIXP0

performed after assigning a new value to $ml(i)$ in the relaxation step, and the ml-value assigned in line 12 is always legal, the first condition holds. Consider now two polygons P_i and P_j with arc (u_i, u_j) in G_p. If the second or third condition would be violated between P_i and P_j, we would have $ml(j) > ml(i) + w_{i,j}$. However, for any two adjacent vertices u_i and u_j in set S we have $ml(j) \leq ml(i) + w_{i,j}$. P_i and P_j are in set S at the termination of the algorithm and hence they are positioned correctly with respect to each other. It follows that configuration C' is feasible.

Let $ml_{left}(j) = x_C(P_j) - x_{C_{left}}(P_j)$. By the definition, $x_{C_{left}}(P_j)$ is the leftmost position for polygon P_j among all feasible configuration. Since C' is feasible, we have $ml(j) \leq ml_{left}(j)$ at the termination of the algorithm. (If this would not hold, C_{left} would not be the left-compressed version of configuration of C.) We next show that the invariant $ml(j) \geq ml_{left}(j)$ is satisfied for all P_j's throughout the algorithm. This and the above condition give $ml(j) = ml_{left}(j)$ at the termination of the algorithm and thus $C' = C_{left}$.

The invariant $ml(j) \geq ml_{left}(j)$ is certainly true after initialization. Consider the first point in time at which the invariant is violated during the algorithm; i.e., we have $ml(j) < ml_{left}(j)$ for some polygon P_j. If the invariant is violated by setting $ml(j)$ in relaxation (line 9), then, just after relaxing the arc (u_i, u_j), we have

$$\begin{aligned} ml(i) + w_{i,j} &= ml(j) \\ &< ml_{left}(j) \\ &\leq ml_{left}(i) + w_{i,j} \end{aligned}$$

which implies that $ml(i) < ml_{left}(i)$. But because relaxing arc (u_i, u_j) does not change $ml(i)$, this inequality must have been true just before we relaxed the arc. This contradicts the choice of P_j as the first polygon for which the invariant is violated.

Assume now that the invariant is violated by setting $ml(j)$ after the legality test (in line 12). Before the legality test we have $x_{\mathcal{C}}(P_j) - ml(j) \leq x_{C_{left}}(P_j)$ and, after executing line 12, P_j is positioned to the right of $x_{C_{left}}(P_j)$. Assume that position $x_{C_{left}}(P_j)$ is in the r-st slot of slot list $\mathcal{S}_j$. We then have $l_{j,r} \leq x_{C_{left}}(P_j)$, where $l_{j,r}$ is the left bound of the interval representing the r-st slot. The search for the next slot (in line 11) yields $q = r$ and we thus have

$$\begin{aligned} ml(j) &= x_{\mathcal{C}}(P_j) - l_{j,q} \\ &\geq x_{\mathcal{C}}(P_j) - x_{C_{left}}(P_j) \\ &= ml_{left}(P_j) \end{aligned}$$

This contradicts our assumption. Hence, the invariant is maintained during algorithm FIXP0 and the lemma follows. □

We now address the time complexity of algorithm FIXP0. If the algorithm given in Figure 3 is applied to G_p, the worst case running time is bounded by $O(\sigma n \log n)$. This bound is achieved when G_p consists of one strongly connected component, $|S_i| = \Theta(s_i)$, the legality test for polygon P_i fails $|S_i|$ times, and for every new legal position an entire shortest path computation is completed. Recall that setting up the necessary data structures for algorithm FIXP0 requires $O(\sigma \log \sigma + (v+k)\log k + (v+\sigma)\log v)$ time.

Theorem 3.1 *Given configuration $\mathcal{C}$, the polygon graph G_p, component graph G_c, and the slot lists, configuration $\mathcal{C}_{left}$ can be generated in $O(\sigma n \log n)$ time.*

Observe that the single source shortest paths problem in planar graphs can be solved in $O(n\sqrt{\log n})$ time, [4]. No straightforward modification of Frederickson's algorithm reduces the running time of algorithm FIXP0 to $O(\sigma n\sqrt{\log n})$ time. When a legality test fails, algorithm FIXP0 reduces the current estimate of $ml(j)$. Future shortest path computations need to take previously computed ml-values into consideration. This corresponds to introducing an edge between u_0 and u_j in G_p with weight $w_{0,j} = ml(j)$ and hence planarity is destroyed.

The remainder of this section describes possible improvements to algorithm FIXP0 that will speed up the algorithm from a practical point of view. The full power of algorithm FIXP0 is only needed for the strongly connected components of G_p and we can improve the running time for generating $\mathcal{C}_{left}$ by handling the strongly connected components of G_p independently. The approach of handling strongly connected components independently has also been used in other compaction problems, [7]. Recall that G_c is the component graph of G_p in which one vertex corresponds to one strongly connected component of G_p. We traverse G_c in a topological order and apply algorithm FIXP0 to the strongly connected components of G_p.

Assume that we have left-compressed the polygons corresponding to the vertices in components $C_1, C_2, \ldots, C_{i-1}$ and we are ready to left-compress the polygons in C_i. Let $G_{p,i}$ be the graph consisting of the subgraph of G_p corresponding to the vertices in C_i and a special vertex u_0. The weights of the edges between two vertices in C_i are set with respect to configuration $\mathcal{C}$. The edges incident to u_0 and their weights are set as follows. Let V_0 be the set of the vertices of G_p in $C_1, C_2, \ldots, C_{i-1}$. If $(u_a, u_b) \in A_p$, where $u_a \in V_0$ and $u_b \in C_i$, set $w_{a,b} \leftarrow w_{a,b} + ml(a)$. The new weight $w_{a,b}$ reflects the maximum amount polygon P_b can move to the left taking into account the positions of the already left-compressed polygons. We shrink the vertices in V_0 to form vertex u_0. If there exists an arc $(u_a, u_b) \in A_p$ with $u_a \in V_0$ and $u_b \in C_i$, we include the edge (u_0, u_b) in $G_{p,i}$. The weight of this arc is set to the minimum over all the newly computed weights of the arcs (u_a, u_b). We then run FIXP0 on $G_{p,i}$. When G_p contains α strongly connected components, with strongly connected component C_i consisting of n_i vertices, configuration $\mathcal{C}_{left}$ is generated in $O(\sum_{i=1}^{\alpha} \sigma_i n_i \log n_i)$ time, where $\sigma_i = \sum_{u_j \in C_i} s_j$. The book-keeping necessary to set up the graphs $G_{p,i}$, $1 \leq i \leq \alpha$, can be accomplished in $O(n)$ total time. This is done by setting the weights $w_{a,b}$ of $G_{p,i}$ while polygon P_a is left-compressed.

Before concluding this section we state the running time for a commonly occurring shape of the polygons. When every polygon is a horizontally convex polygon (i.e., no horizontal line intersects the polygon more than once), the running time for determining configuration $\mathcal{C}_{left}$ reduces to $O(\sigma)$. In this case we have $G_p = G_c$, each strongly connected component contains only one polygon. Ignoring the time needed to determine the next slot in case a legality test fails, algorithm FIXP0 handles one component in $O(1)$ time. Since $|S_i| \geq 1$, we have $\sigma \geq n$. For the case when every polygon is a rectangle this running time has previously been reported in [6].

4 Forbidden region problem

In this section we present two algorithms for determining a minimum configuration for the forbidden region problem. Both algorithms use a characterization of the left-compressed minimum configuration that allows us to limit the number of left-compressed configurations to be considered.

Assume the layout area contains no forbidden regions. Let $\mathcal{C}^*$ be the left-compressed configuration that has polygon P_0 positioned at the origin in this environment (namely, the one without forbidden regions). As already stated in Section 3, $\mathcal{C}^*$ can be determined in $O(n \log n)$ time by performing a shortest path computation on G_p. Using the algorithm of [4], $\mathcal{C}^*$ can actually be determined in $O(n\sqrt{\log n})$ time. The position of polygon P_i in configuration $\mathcal{C}^*$, $x_{\mathcal{C}^*}(P_i)$, is a crucial quantity in our forbidden region algorithms. We will also be referring to the positions of the vertical edges of polygons and thus let $x_{\mathcal{C}^*}(e_{i,q})$ be the position of edge $e_{i,q}$ of P_i in configuration $\mathcal{C}^*$. Let $\mathcal{C}$ be any feasible configuration in which P_0 is positioned to the left of all forbidden regions. We say polygon P_i (resp. edge $e_{i,q}$ of P_i) is at its *tight* position in $\mathcal{C}$ if $x_{\mathcal{C}}(P_i) = x_{\mathcal{C}}(P_0) + x_{\mathcal{C}^*}(P_i)$ (resp. $x_{\mathcal{C}}(e_{i,q}) = x_{\mathcal{C}}(P_0) + x_{\mathcal{C}^*}(e_{i,q})$). Intuitively, P_i is at its tight position if the position of P_i is not influenced by the presence of any forbidden region.

Observe that placing P_i at a position to the left of P_i's tight position results in a configuration that is not feasible.

The following lemma characterizes a left-compressed configuration of minimum width. Recall that an edge of forbidden region B_j and an edge $e_{i,q}$ of polygon P_i are related if there exists a horizontal line that intersects both edges.

Lemma 4.1 *There exists a left-compressed, minimum configuration $\hat{C}$, a polygon P_i and a forbidden region B_j, $0 \leq i \leq n$, $1 \leq j \leq k$, such that*

$$x_{\hat{C}}(P_0) = x(B_j) - x_{C^*}(e_{i,q})$$

for some vertical edge $e_{i,q}$ of P_i related to B_j.

Proof: Let G_p be the polygon graph in which the weights are set with respect to configuration $\hat{C}$. From G_p we generate a new graph G'_p by adding to G_p a new vertex u_B and the following arcs. Let e_j be the left vertical edge of forbidden region B_j and let $e_{i,q}$ be a vertical edge of a polygon P_i. If e_j is visible from $e_{i,q}$ in configuration $\hat{C}$, G'_p contains the arc (u_i, u_B). The weight of the arc (u_i, u_B) in G'_p is the minimum over all the distances between a left vertical edge e_j of a forbidden region and a vertical edge $e_{i,q}$ of polygon P_i with e_j visible from $e_{i,q}$. Obviously, if there exists a left vertical edge $e_{i,q}$ with $x_{\hat{C}}(e_{i,q}) = x(B_j)$, for some forbidden region B_j, then the weight of arc (u_i, u_B) is zero.

Let $l(u)$ be the length of the shortest path from vertex u_0 to any vertex u in G'_p. When $l(u_B) = 0$, the property stated in the lemma holds in configuration $\hat{C}$. If $l(u_B) \neq 0$, we generate a configuration C' from $\hat{C}$ by pushing polygon P_0 $l(u_B)$ positions to the right. During this pushing process, P_0 may touch some other polygon. If this happens, this polygon is pushed to the right to along with P_0. At some later point, another polygon may be touched either by P_0 or by the polygon that already moves with P_0. This new polygon is also pushed along. This process continues until P_0 has been pushed $l(u_B)$ positions. At this

point either a polygon P_i that is being pushed or P_0 touch a forbidden region, say B_j. Let C' be the configuration generated by the pushing process. Since we assumed $\hat{C}$ to be a minimum configuration, the width of configuration C' cannot be smaller than that of $\hat{C}$. This implies that the pushing process resulted in moving P_{n+1} $l(u_B)$ positions to the right. The claimed property now holds for minimum configuration C'. □

Each related pair $(B_j, e_{i,q})$ induces a possible position for P_0. Lemma 4.1 states that a minimum, left-compressed configuration is among the σ left-compressed configurations induced by the σ related pairs. The proof of Lemma 4.1 gives us additional information about the related pair inducing the position of P_0 in configuration $\hat{C}$. We can conclude that if $\hat{C}$ is a minimum, left-compressed configuration, then there not only exist a P_i and a B_j with $x_{\hat{C}}(P_0) = x(B_j) - x_{C^*}(e_{i,q})$, but that the position of edge $e_{i,q}$ is tight. We thus have $x_{\hat{C}}(e_{i,q}) = x(B_j)$ (i.e., the edge $e_{i,q}$ is immediately to the left of forbidden region B_j).

In Section 3 we described algorithm FIXP0 which generates the left-compressed minimum configuration when the position of P_0 is fixed. We solve the forbidden region problem by invoking algorithm FIXP0 σ times, once for every position induced by a related pair $(B_j, e_{i,q})$. However, by invoking FIXP0 in the right order we are able to achieve a running time that equals the worst-case running time of a single application of algorithm FIXP0.

The positions of polygon P_0 in the σ configurations can be determined by using information available in the slot lists. Let $\mathcal{P}$ be the list containing these σ positions arranged by increasing x-values. List $\mathcal{P}$ is generated in $O(\sigma \log \sigma)$ time. Let $C_1, C_2, \ldots, C_\sigma$ be the left-compressed configurations associated the possible positions of P_0 with $x_{C_a}(P_0) < x_{C_{a+1}}(P_0)$, $1 \leq a \leq \sigma - 1$. Observe that the actual number of configurations to be considered is likely to be smaller than σ. This holds since not every related pair does necessarily introduce a legal right-value and because we can discard from list $\mathcal{P}$ entries which correspond

to positions of P_0 to the right of the leftmost position of any forbidden region. However, to simplify notation, we let σ be the number of configurations to be considered. When the configurations are generated in the order $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\sigma$, we have $x_{\mathcal{C}_a}(P_i) \leq x_{\mathcal{C}_{a+1}}(P_i)$, $0 \leq i \leq n+1$ (i.e., every polygon can only move to the right).

Let $\mathcal{C}$ be the given feasible configuration to be compacted. We assume that in configuration $\mathcal{C}$ P_0 is positioned to the left and all other polygons are positioned to the right of the forbidden regions. However, this holds w.l.o.g and any feasible configuration could be used. Both of our algorithms assume that the polygon graph G_p, the component graph G_c, and the slot lists have already been determined. The weights of G_p are set with respect to configuration $\mathcal{C}$. Our first algorithm, algorithm GENCONF1, determines the position of P_0 resulting in the left-compressed, minimum configuration by invoking FIXP0 σ times. The order of the positions of P_0 considered is as given by list $\mathcal{P}$. Generating the σ configurations in this order allows us to determine a minimum configuration in $O(\sigma n \log n)$ time. Recall that one application of algorithm FIXP0 has a worst-case running time of $O(\sigma n \log n)$. Having the worst-case running time of algorithm FIXP0 coincide with the time required to determine the width of all σ configurations holds only for non-convex polygons. For convex polygons FIXP0 runs in $O(\sigma)$ time and algorithm GENCONF1 can be shown to use $O(\sigma n)$ time. Our second algorithm, algorithm GENCONF2, determines the width associated with every configuration $\mathcal{C}_a$ by limiting the number of polygons that need to be repositioned explicitly. While its the worst-case running time is as for GENCONF1, we are able to state the running time in a more problem-dependent form, as will be described later. We now give the details for algorithm GENCONF1.

For algorithm GENCONF1 we can eliminate from list $\mathcal{P}$ entries with identical positions for P_0. Let $G_p(\mathcal{C}_a)$ be the polygon graph associated with configuration $\mathcal{C}_a$. $G_p(\mathcal{C}_a)$ differs from G_p only in the value of the weights associated with the

Algorithm **GENCONF1:**
Input: Graph G_p with weights set with respect to initial configuration $\mathcal{C}$
Output: *mina*, the index resulting in a minimum configuration

1. $minwidth \leftarrow +\infty$;
2. **for** $a := 1$ **to** σ **do**
   begin
      (* generate $G_p(\mathcal{C}_a)$ from $G_p(\mathcal{C}_{a-1})$ *)
      (* $\mathcal{C}_0$ corresponds to the initial configuration $\mathcal{C}$ *)
3.   **for** each vertex u_j such that $(u_0, u_j) \in A_p$ **do**
4. $w_{0,j}(G_p(\mathcal{C}_a)) \leftarrow w_{0,j}(G_p(\mathcal{C}_{a-1})) - (x_{\mathcal{C}_a}(P_0) - x_{\mathcal{C}_{a-1}}(P_0))$;
5.   FIXP0($G_p(\mathcal{C}_a)$);
6.   **if** $minwidth > x_{\mathcal{C}_a}(P_{n+1}) - x_{\mathcal{C}_a}(P_0)$ **then**
      begin
7. $minwidth \leftarrow x_{\mathcal{C}_a}(P_{n+1}) - x_{\mathcal{C}_a}(P_0)$;
8. $mina \leftarrow a$;
      end (* then *)
   end (* for *)

Figure 4: Algorithm GENCONF1

edges incident to u_0. Recall that the weight of the arc (u_i, u_j) in G_p reflects the maximum amount polygon P_j can move to the left without overlapping with polygon P_i when P_i is kept fixed and the forbidden regions are not taken into account. The weight of arc (u_0, u_j) in graph $G_p(\mathcal{C}_a)$ reflects the maximum distance P_j could move to the left before it hits the position of P_0 in configuration $\mathcal{C}_a$. Hence, in $G_p(\mathcal{C}_a)$ we have $w_{0,j}(G_p(\mathcal{C}_a)) = w_{0,j}(G_p) - (x_{\mathcal{C}_a}(P_0) - x_{\mathcal{C}}(P_0))$. Figure 4 gives a detailed description of algorithm GENCONF1.

The correctness of algorithm GENCONF1 follows immediately and we now address its time complexity. The minimum amount of work done in one instance of FIXP0 is $O(n \log n)$ and thus algorithm GENCONF1 uses at least $O(\sigma n \log n)$ total time. Any additional time used during algorithm GENCONF1 comes from having legality tests fail. As described in Section 3, the failure of one legality test can cause an entire shortest path computation to be completed before the

next legality test fails. During the generation of the σ configurations polygons moves to the right and thus polygon P_i causes the failure of at most s_i legality test. Hence, the additional time spent during the σ calls to FIXP0 is bounded by $O(\sigma n \log n)$.

Theorem 4.1 *Given the polygon graph G_p, component graph G_c, and the slot lists, a minimum, left-compressed configuration of a given feasible configuration can be generated in $O(\sigma \log \sigma + \sigma n \log n)$ time.*

Algorithm GENCONF1 can be speeded up by using the improved version of FIXP0 in which the strongly connected components of G_p are handled independently. Using this version of FIXP0, the running time of algorithm GENCONF1 is bounded by $O(\sigma(\sum_{i=1}^{\alpha} n_i \log n_i))$, where α is the number of strongly connected components in G_p. In the worst case, this time bound is equal to the time bound stated in the theorem above.

The way algorithm GENCONF1 sets the weights of the arcs incident to vertex u_0 ensures that some re-computations are avoided. However, when going from configuration $\mathcal{C}_{a-1}$ to configuration $\mathcal{C}_a$ we only need to determine the position of polygon P_{n+1} in $\mathcal{C}_a$ (since its position gives us the width associated with configuration $\mathcal{C}_a$). Once the index resulting in a left-compressed minimum configuration is known, algorithm FIXP0 can be called to determine the position of the polygons in the minimum configuration. Algorithm GENCONF2 determines the width associated with configuration $\mathcal{C}_a$ by making use of information computed by configuration $\mathcal{C}_{a-1}$ and it limits the number of new positions to be determined explicitly. For the time being assume that every related pair induces a unique position for polygon P_0 (the changes to be made when this is not satisfied are described later).

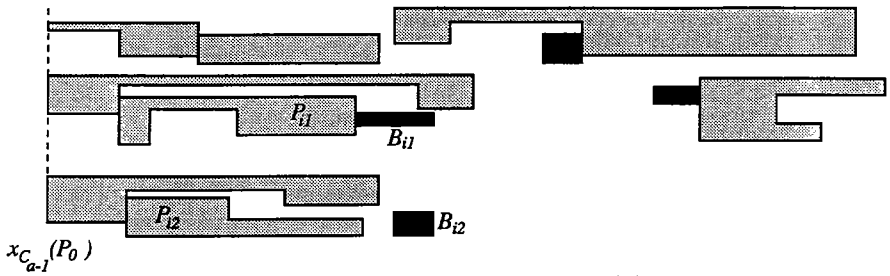
Let edge $e_{i1,q1}$ of polygon P_{i1} and forbidden region B_{j1} be the related pair dictating the position of P_0 in $\mathcal{C}_{a-1}$ (i.e., $x_{\mathcal{C}_{a-1}}(P_0) = x(B_{j1}) - x_{C^*}(e_{i1,q1})$). Let

$e_{i2,q2}$ and B_{j2} be the related pair dictating the position of P_0 in $\mathcal{C}_a$. Also, let $\epsilon_a = x_{\mathcal{C}_a}(P_0) - x_{\mathcal{C}_{a-1}}(P_0)$. Intuitively, in order to generate configuration $\mathcal{C}_a$ from $\mathcal{C}_{a-1}$, polygon P_0 is pushed ϵ_a positions to the right. A polygon that is touched by P_0 during this pushing process is pushed along with it. When a polygon already being pushed touches other polygon, this polygon is also pushed along. This pushing process can push polygons anywhere from 0 to ϵ_a positions to the right. Assume that in configuration $\mathcal{C}_{a-1}$ edge $e_{i1,q1}$ of polygon P_{i1} is to the left of forbidden region B_{j1} and that edge $e_{i2,q2}$ of polygon P_{i2} is to the left of forbidden region B_{j2}. Such a situation is shown in Figure 5(a). After the pushing process is completed, edge $e_{i2,q2}$ is immediately to the left of forbidden region B_{j2} and edge $e_{i1,q1}$ of polygon P_{i1} is to the right of the left vertical edge of forbidden region B_{j1}, possibly overlapping with B_{j1}. Figure 5(b) illustrates the situation after the pushing process. At this point polygon P_{i1} and all polygons reachable from P_{i1} may need new legal positions. We say polygon P_t is *reachable* from P_{i1} if there exists a path from u_{i1} to u_t in G_p. Figure 5(c) shows configuration $\mathcal{C}_a$ in which polygon P_{i1} and all polygons reachable from it have new positions (if needed).

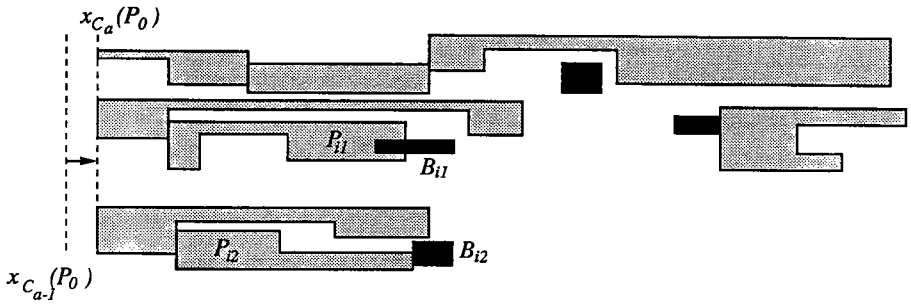
We next prove a lemma which will allow us to avoid determining the positions for polygons not reachable by P_{i1}.

Lemma 4.2 *Let P_t be a polygon not reachable by P_{i1}. Then, the pushing process initiated by P_0 moves P_t at most ϵ_a positions to the right. During the pushing process P_t does not move across a left vertical edge of a forbidden region. If P_t is moved at least one position to the right, then, after the pushing process, P_t is at its tight position.*

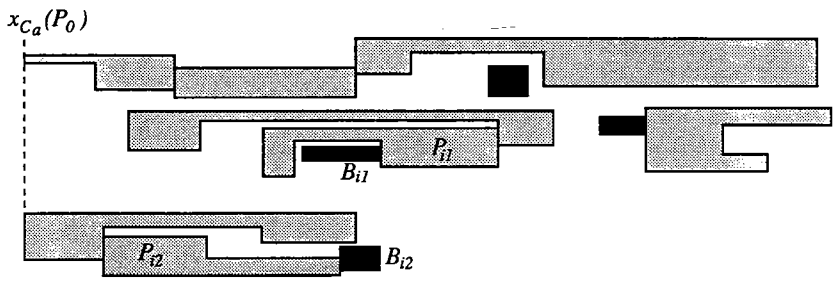
Proof: Assume that during the pushing process P_t is pushed across the left vertical edge of forbidden region B_s. Let e_s be this left vertical edge. Then, there must exist a vertical edge $e_{t,q}$ of P_t such that the distance between $e_{t,q}$ and



(a) Configuration $\mathcal{C}_{a-1}$



(b) After P_0 is pushed ϵ_a positions to the right



(c) Configuration $\mathcal{C}_a$

Figure 5: Generating $\mathcal{C}_a$ from $\mathcal{C}_{a-1}$

e_s is less than ϵ_a in $\mathcal{C}_{a-1}$; i.e., $x_{\mathcal{C}_{a-1}}(e_s) - x_{\mathcal{C}_{a-1}}(e_{t,q}) < \epsilon_a$. This implies that there exists a configuration $\mathcal{C}'$ dictated by $e_{t,q}$ and B_s with $x_{\mathcal{C}'}(P_0) = x(B_s) - x_{\mathcal{C}^*}(e_{t,q})$ such that $x_{\mathcal{C}_{a-1}}(P_0) < x_{\mathcal{C}'}(P_0) < x_{\mathcal{C}_a}(P_0)$. Such a configuration $\mathcal{C}'$ cannot exist and thus P_t cannot move across edge e_s. Assume now that polygon P_t is moved at least one position to the right. Consider the visibility graph in which the weights are set with respect to the configuration generated after the pushing process. In this visibility graph the shortest path from u_0 to u_t has length 0 and thus P_t is at a tight position. □

Algorithm GENCONF2 determines the width of configuration $\mathcal{C}_a$ by computing new positions only for polygons reachable from P_{i1}. For polygons not reachable from P_{i1} no updating of the positions is done. However, should we need to determine the position of a polygon (or an edge of a polygon), we can do so in $O(1)$ time. Assume we have handled configuration $\mathcal{C}_{a-1}$. For a polygon P_t that is not at its tight position in $\mathcal{C}_{a-1}$ $ml(t)$ gives the amount P_t moves to the left from its position in configuration $\mathcal{C}$ to its position in $\mathcal{C}_{a-1}$. Hence, the position of any polygon P_t in $\mathcal{C}_{a-1}$ is determined by

$$x_{\mathcal{C}_{a-1}}(P_t) = \max\{x_{\mathcal{C}_{a-1}}(P_0) + x_{\mathcal{C}^*}(P_t), x_{\mathcal{C}}(P_t) - ml(t)\}.$$

Observe that the quantity $x_{\mathcal{C}_{a-1}}(P_0) + x_{\mathcal{C}^*}(P_t)$ corresponds to the tight (i.e., leftmost possible) position of polygon P_t in configuration $\mathcal{C}_{a-1}$.

We are now ready to give a complete description of algorithm GENCONF2. The preprocessing step again includes computing the σ x-positions of polygon P_0 and arranging these positions according to non-decreasing x-values. Assume further we determined for every polygon P_i a list R_i containing the polygons reachable from P_i (which includes P_i). The input to algorithm GENCONF2 is the polygon graph G_p with its weights set with respect to the initial configuration $\mathcal{C}$. The output is the index of the configuration resulting in the left-compressed minimum configuration. Figure 6 contains a detailed description of algorithm GENCONF2 and the following discussion refers to the steps of this description.

Algorithm **GENCONF2**;
Input: Graph G_p with weights set with respect to configuration $\mathcal{C}$;
Output: *mina*, the index resulting in a minimum configuration;

(* Determine the *ml*-values of configuration $\mathcal{C}_1$ *)
1. call FIXP0($G_p(\mathcal{C}_1)$);
2. $minwidth \leftarrow x_{\mathcal{C}_1}(P_{n+1}) - x_{\mathcal{C}_1}(P_0)$; $mina \leftarrow 1$;
3. **for** $a := 2$ **to** σ **do**
begin

(* Determine whether the position of P_{i1} is tight in $\mathcal{C}_{a-1}$ *)
(* If it is, move P_{i1} ϵ_a positions to the right; perform a legality *)
(* test and update *ml*-values for polygons reachable from P_{i1} *)
4. **if** the position of edge $e_{i1,q1}$ in $\mathcal{C}_{a-1}$ is tight **then**
begin
5. $ml(i1) \leftarrow x_{\mathcal{C}}(P_{i1}) - (x_{\mathcal{C}_a}(P_0) + x_{\mathcal{C}^*}(P_{i1}))$;
6. call LEGAL($i1$);
7. call UPDATE_ML(G_p, R_{i1});
end (* then *)

8. **if** the position of edge $e_{i2,q2}$ in $\mathcal{C}_a$ is tight **then**
begin
(* Determine the width of configuration $\mathcal{C}_a$ *)
9. $x_{\mathcal{C}_a}(P_{n+1}) = \max\{x_{\mathcal{C}_a}(P_0) + x_{\mathcal{C}^*}(P_{n+1}), x_{\mathcal{C}}(P_{n+1}) - ml(n+1)\}$;
10. **if** $minwidth > x_{\mathcal{C}_a}(P_{n+1}) - x_{\mathcal{C}_a}(P_0)$
then $minwidth \leftarrow x_{\mathcal{C}_a}(P_{n+1}) - x_{\mathcal{C}_a}(P_0)$; $mina \leftarrow a$;
end (* then *)
end (* for *)

Figure 6: Algorithm GENCONF2

Algorithm **UPDATE_ML**;
Input: Set R_{i1} and G_p with weights set with respect to configuration $\mathcal{C}$;
Output: No output is generated; global array ml is modified;

```
1.    for all u_t ∈ R_i1 − {u_i1} do ml(t) ← +∞;
2.    S ← ∅;
3.    Q ← R_i1;
4.    while Q ≠ ∅ do
      begin
5.      u_i ← ExtractMin(Q);
6.      S ← S ∪ {u_i};
7.      for each vertex u_j such that (u_i, u_j) ∈ A_p do
        begin
8.        if ml(j) > ml(i) + w_{i,j} then
          begin
9.          ml(j) ← ml(i) + w_{i,j};
10.         call LEGAL(j);
            (* Lower bound checking for the position of P_j *)
11.         if x_C(P_j) − ml(j) < x_{C_a}(P_0) + x_{C*}(P_j)
            then ml(j) ← x_C(P_j) − (x_{C_a}(P_0) + x_{C*}(P_j));
12.         if u_j ∈ S then S ← S − {u_j}; Q ← Q ∪ {u_j};
          end(* then *)
      end (* for u_j *)
    end (* while Q *)
```

Figure 7: Algorithm UPDATE_ML

The first step of algorithm GENCONF2 makes a call to FIXP0 to determine the ml-values resulting in the positions of the polygons in configuration $\mathcal{C}_1$. Obviously, using the above expression for computing positions in a configuration, gives the correct positions of the polygons in $\mathcal{C}_1$. Assume we have handled configuration $\mathcal{C}_{a-1}$ and we are to determine the width of configuration $\mathcal{C}_a$. We first check whether the position of edge $e_{i1,q1}$ is tight. If it is, then edge $e_{i1,q1}$ is immediately to the left of forbidden region B_{j1} and we move polygon P_{i1} ϵ_a positions to the right. This may put P_{i1} at a position that is not legal and we perform a legality test on this new position (actually, on the ml-value associated with this position). Procedure LEGAL($i1$) used by GENCONF2 tests whether $ml(i1)$ corresponds to a legal position for P_{i1} and, if it does not, it determines the closest legal position to the right of it (it basically executes lines 10-12 of algorithm FIXP0).

As already discussed, we only update the ml-values to correspond to correct ml-values for configuration $\mathcal{C}_a$ for polygons reachable from P_{i1}. These ml-values are generated by running on set R_{i1} an algorithm similar to FIXP0. We call this algorithm UPDATE_ML. For clarity reasons and, since there are some differences between FIXP0 and UPDATE_ML, we give its entire description in Figure 7. The input for UPDATE_ML are G_p and R_{i1}. We initialize the priority Q with $Q \leftarrow R_{i1}$ and set $ml(t) \leftarrow +\infty$ for all $u_t \in R_{i1} - \{u_{i1}\}$. We then determine the ml-values of the polygons in R_{i1} by performing relaxation steps and legality tests. Whenever $ml(t)$ gets a new value by either relaxation or legality test, we have to perform an additional test on $ml(t)$, namely a lower bound test. Recall that in configuration $\mathcal{C}_a$ the position of P_t cannot be to the left of its tight position (i.e., it cannot be smaller than $x_{\mathcal{C}_a}(P_0) + x_{\mathcal{C}^*}(P_j)$). However, since UPDATE_ML is performed only on polygons reachable from P_{i1}, relaxation and legality tests can produce an ml-value corresponding to a position to the left of the tight position. The lower bound test checks and, when necessary, corrects

the ml-values.

When edge $e_{i1,q1}$ is not at its tight position in configuration $\mathcal{C}_{a-1}$, P_{i1} is to the right of forbidden region B_{j1} and the value of $ml(i1)$ gives the position of P_{i1} in $\mathcal{C}_{a-1}$. The process of pushing P_0 ϵ_a positions to the right may leave P_{i1} at the same position or it may move it to the right. If it does move P_{i1} to the right, then P_{i1} is at its tight position in $\mathcal{C}_a$ and its ml-value does no longer determines its position. Neither polygon P_{i1} nor any polygon reachable from it require that a new legal position is determined explicitly.

After having executed steps 1-7 of algorithm GENCONF2 we have determined enough information about configuration $\mathcal{C}_a$ so that we can compute its width. We first check whether the previous configurations or the process of re-positioning the polygons reachable from P_{i1} resulted in having edge $e_{i2,q2}$ of polygon P_{i2} be to the right of its tight position in $\mathcal{C}_a$. If the edge $e_{i2,q2}$ is already to the right of forbidden region B_{j2}, then we can immediately say that the position of P_0 induced by the related pair $(e_{i2,q2}, B_{j2})$ cannot result in a minimum configuration. If the position of edge $e_{i2,q2}$ is tight, we determine the width of configuration $\mathcal{C}_a$.

The description of algorithm GENCONF2 given assumes that every related pair induces a unique position for P_0. We briefly describe the changes to be made if this is not the case. Assume there exist l_1 related pairs that induce the $(a-1)$-st position of polygon P_0. In order to determine the width of the configuration induced by the a-th position of P_0, we perform steps 4-7 for every related pair inducing the $(a-1)$-st position of polygon P_0. Assume now that there are l_2 related pairs that induce the a-st position of polygon P_0. Then, as long as one polygon of the l_2 polygons has its vertical edge at a tight position, configuration $\mathcal{C}_a$ could be a minimum configuration. We determine the width of the configuration as described in lines 9 and 10 of algorithm GENCONF2.

We now establish the time bound of algorithm GENCONF2. Let $r_i = |R_i|$,

the number of polygons which can be reached by polygon P_i, $r_i \geq 1$. Each time an edge $e_{i1,q1}$ of polygon P_{i1} is pushed across forbidden region B_{j1}, we update the *ml*-entries of all the polygons reachable from P_{i1}. This is done by procedure UPDATE_ML and costs at least $O(r_i \log r_i)$ time. A polygon P_i is pushed across at most s_i forbidden regions and every such event may invoke a call to UPDATE_ML. Let $\Delta = \sum_{i=1}^{n} s_i r_i$, $\Delta \leq \sigma n$. The time required by GENCONF2 is thus at least $O(\sum_{i=1}^{n} s_i r_i \log r_i) = O(\Delta \log n)$. Consider now the additional time spent in procedure UPDATE_ML because legality tests fail. Assume we called UPDATE_ML with set R_i and that a legality test fails for some polygon P_j, $P_j \in R_i$. Then, the amount of time that elapses before the next legality test fails is $O(r_j \log r_i)$. We charge this time to polygon P_j. Polygon P_j is charged at most $O(s_j r_j \log n)$ time and the time charged to all polygons is thus bounded by $O(\sum_{j=1}^{n} s_j r_j \log n) = O(\Delta \log n)$. Traversing the slot lists during algorithm GENCONF2 costs $O(\sigma)$ time which is less than $O(\Delta \log n)$. We can thus state the following theorem.

Theorem 4.2 *Given the polygon graph G_p, component graph G_c, and the slot lists, a minimum, left-compressed configuration of a given feasible configuration can be generated in $O(\sigma \log \sigma + \Delta \log n)$ time.*

5 k-partition problem

In the k-partition problem the height of every forbidden region equals h, the height of the layout area. Hence, the forbidden regions model positions in the layout area where the layout could be "cut" by a straight vertical line. Obviously, the algorithms described in the previous section can be used to generate a minimum configuration for the k-partition problem. In this section we describe a more efficient approach which is tailored towards the special structure of the forbidden regions. We present an algorithm determining a minimum configura-

tion in $O(\delta(\log n + \log \lceil \frac{\alpha k}{\rho} \rceil))$ time, where δ and ρ are two quantities measuring reachability in the visibility graphs, $\delta \leq \alpha n$ and $\rho \leq \alpha^2$. The preprocessing time of the algorithm is $O(k \log k + v \log v + \rho)$.

Assume that the width of every forbidden region is zero. Straightforward modifications to the algorithm can handle forbidden regions with arbitrary widths. Assume, furthermore, that the k forbidden regions have been ordered so that $x(B_1) < x(B_2) < \ldots < x(B_k)$. In the general forbidden region problem every polygon P_i has a slot list $\mathcal{S}_i$ associated with it and $\mathcal{S}_i$ records the forbidden regions relevant to P_i. In the k-partition problem we have one data structure, namely a level-linked finger search tree [8], which is accessed by all n polygons. Let A_i be the area, called *slot area*, between forbidden regions B_i and B_{i+1}, with A_0 being the area to the left of B_1 and A_k being the area the right of B_k, respectively. Slot area A_i is represented by the interval $[x(B_i), x(B_{i+1})]$ for $1 \leq i \leq k-1$. When $i = 0$ and k, the intervals are $(-\infty, x(B_1)]$ and $[x(B_k), +\infty)$, respectively. In the level-linked finger search tree used by our algorithm the leaves correspond to the slot areas $A_0, A_1, \ldots, A_k$.

Since the height of every forbidden region is h, the following property follows immediately.

Property 5.1 *Let C be a feasible configuration in which polygon P_i is assigned a position in slot area A_q. Then, every polygon P_j in the same strongly connected component with P_i in G_p is also assigned a position in slot area A_q.*

As defined in Section 4, let C^* be the left-compressed configuration that has polygon P_0 positioned at the origin in a layout environment without forbidden regions. Let C_i be a strongly connected component of G_p. The *rightmost* vertical edge of component C_i in configuration C^* is the rightmost position of any edge in C^* belonging to a polygon in C_i. Let $x_{C^*}(C_i)$ be this position. Using Property 5.1 and adjusting Lemma 4.1 to the k-partition problem, the next lemma characterizes a minimum, left-compressed configuration.

Lemma 5.1 *There exists a left-compressed, minimum configuration $\hat{C}$ and a strongly connected component C_i of G_p such that*

$$x_{\hat{C}}(P_0) = x(B_1) - x_{C^*}(C_i).$$

Lemma 5.1 implies that every strongly connected component of G_p induces a possible position for P_0. The proof of Lemma 5.1 is similar to that of Lemma 4.1 and is omitted. Since G_p contains α strongly connected component, we need to determine the width of α left-compressed configurations. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\alpha$ be the left-compressed configurations associated the possible positions of P_0 with $x_{\mathcal{C}_a}(P_0) < x_{\mathcal{C}_{a+1}}(P_0)$, $1 \leq a \leq \alpha - 1$. Our algorithm generates the configurations in the order of $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\alpha$. For convenience assume the position of P_0 in configuration $\mathcal{C}_a$ is determined by the rightmost vertical edge of strongly connected component C_a.

Configuration $\mathcal{C}_1$ is obtained by positioning P_0 at $x_{\mathcal{C}_1}(P_0)$ and left-compressing all polygons, except P_{n+1}, into the first slot area A_0. To generate configuration $\mathcal{C}_a$ from $\mathcal{C}_{a-1}$, intuitively, we push P_0 from position $x_{\mathcal{C}_{a-1}}(P_0)$ to position $x_{\mathcal{C}_a}(P_0)$. Let $\epsilon_a = x_{\mathcal{C}_a}(P_0) - x_{\mathcal{C}_{a-1}}(P_0)$. This pushing can effect a polygon P_i in one of three ways.

Case 1. In configuration $\mathcal{C}_{a-1}$ polygon P_i is in slot area A_0 and P_i is not reachable from the polygons in component C_{a-1}. In this case polygon P_i is at its tight position in configuration $\mathcal{C}_{a-1}$. The pushing shifts polygon P_i ϵ_a positions to the right and P_i is at its tight position in configuration $\mathcal{C}_a$.

Case 2. In configuration $\mathcal{C}_{a-1}$ polygon P_i is not in slot area A_0 and P_i not reachable from the polygons in component C_{a-1}. The pushing does not affect polygon P_i. Thus, $x_{\mathcal{C}_a}(P_i) = x_{\mathcal{C}_{a-1}}(P_i)$.

Case 3. Polygon P_i is reachable from the polygons in component C_{a-1}. The pushing may force polygon P_i to move to the right. In this case we re-compute the position of P_i.

We determine the width of configuration $\mathcal{C}_a$ by re-computing the positions of polygons reachable from the polygons in component C_{a-1}. By Property 5.1, we know that the polygons in the same strongly connected component are assigned to positions in the same slot area. A basic step of the algorithm is the left-compression of a given strongly connected component. When generating configuration $\mathcal{C}_a$ from $\mathcal{C}_{a-1}$ we left-compress the strongly connected components reachable from the polygons in component C_{a-1} in a topological order induced by G_c. For strongly connected component C_i let R_i be its *reachability list.* R_i contains all polygons reachable from the polygons in component C_i. The polygons in R_i are grouped according to the strongly connected components they belong to and these components are arranged in a topological order induced by G_c.

We now describe how to generate configuration $\mathcal{C}_a$ from $\mathcal{C}_{a-1}$. In configuration $\mathcal{C}_{a-1}$ the rightmost vertical edge of strongly connected component C_{a-1} is at position $x(B_1)$. In configuration $\mathcal{C}_a$, the polygons of component C_{a-1} are assigned to a slot area to the right of B_1. We perform a left-compression of the polygons in C_{a-1} against a straight vertical edge and determine the width needed to accommodate these polygons. We then determine the leftmost slot area that can accommodate the polygons in component C_{a-1}. Let A_s be this slot area, $s \geq 1$. Once the polygons in component C_{a-1} have been placed into slot area A_s, the components in reachability list R_{a-1} are handled. Every component in list R_{a-1} may need to be left-compressed.

Assume that we have determined the new positions for the polygons in the first $i-1$ components in reachability list R_{a-1}. Let C_t be the i-th component in R_{a-1}. Every polygon in component C_{a-1} and every polygon in a component C_j, $j \neq t$, that can reach the polygons in C_t has been assigned a new position. Let q' be the largest index of a slot area containing any of these polygons. The polygons in component C_t are now either assigned positions in slot area $A_{q'}$ or

in a slot area to the right of $A_{q'}$. We first perform a left-compression of the polygons in C_t into slot area $A_{q'}$. How far to the left a polygon P_s in C_t can be positioned in slot area $A_{q'}$ depends on the positions of polygons that can reach P_s. If the new position of any polygon of C_t causes an overlap with forbidden region $B_{q'+1}$, the polygons in component C_t cannot be placed into slot area $A_{q'}$. In this case we left-compress the polygons in C_t against a straight vertical edge and determine the closest slot area that can accommodate component C_t. Let A_q be this slot area. After we have processed all polygons in R_{a-1}, the width of configuration $\mathcal{C}_a$ is determined. The book-keeping necessary to perform the left-compression of the polygons in a component is similar to the book-keeping done in algorithm GENCONF2 described in the previous section.

We now establish the time bound of our algorithm for k-partition problem. The preprocessing steps include setting up a level-linked finger search tree for $k+1$ slot areas, generating the visibility graphs G_p and G_c, and determining the reachability lists. Let rc_i be the number of strongly components of G_p that can reach component C_i and let $\rho = \sum_{i=0}^{\alpha} rc_i$, $\rho \leq \alpha^2$. The total preprocessing time is $O(k \log k + v \log v + \rho)$.

Let $rp_a = |R_a|$, the number of polygons reachable by the polygons in strongly connected component C_a, $rp_a \geq 1$. When the polygons in component C_{a-1} are pushed across forbidden region B_1, we update the ml-entries of the polygons in R_{a-1}. When generating configuration $\mathcal{C}_a$ from $\mathcal{C}_{a-1}$, each component C_t reachable from component C_{a-1} is left-compressed at most twice. Ignoring the time needed to determine the slot area A_q (in case C_t does not fit into $A_{q'}$), it takes $O(rp_{a-1} \log rp_{a-1})$ time to update the ml-entries of the polygons in R_{a-1}. Let $\sum_{a=1}^{\alpha} rp_a = \delta$, $\delta < \alpha n$. The time spent on performing left-compression is bounded by $O(\delta \log n)$ time.

Now consider the time needed for determining the slot area A_q. As already stated, the intervals representing the slot areas are the leaves of a level-linked

finger search tree. Initially, a component C_i is to the left of B_1. At some point C_i moves across B_1 and is assigned a slot area to the right of B_1. From this point on, every time a strongly connected component that can reach C_i moves across B_1, we may have to determine a new slot area for C_i. A new slot area is always to the right of an old one. Recall that rc_i represents the number of components that can reach component C_i. Also, let $f_{i,j}$ be the number of slot areas component C_i moves to the right when C_i searches for a new slot area for the j-th time, $1 \leq j \leq rc_i$. By using the level-linked finger search tree, the total time needed to determine the new slot areas for C_i is

$$\sum_{j=1}^{rc_i} (1 + \log f_{i,j})$$

which is less than

$$rc_i \log \frac{k}{rc_i} + rc_i.$$

Let T_p be the total time needed to determine the new slot areas for all components. Then,

$$\begin{aligned} T_p &\leq \sum_{i=0}^{\alpha} \left(rc_i \log \frac{k}{rc_i} + rc_i \right) \\ &\leq \sum_{i=0}^{\alpha} rc_i \log k - \sum_{i=0}^{\alpha} rc_i \log rc_i + \sum_{i=0}^{\alpha} rc_i \\ &\leq \rho \log k - \sum_{i=0}^{n} rc_i \log rc_i + \rho, \end{aligned}$$

where $\sum_{i=0}^{\alpha} rc_i = \rho < \alpha^2$. A straightforward computation shows that

$$\sum_{i=0}^{\alpha} rc_i \log rc_i > \rho \log \frac{\rho}{\alpha}.$$

Using this lower bound on $\sum_{i=0}^{\alpha} rx_i \log rc_i$, we get

$$T_p < \rho \log \left\lceil \frac{\alpha k}{\rho} \right\rceil + \rho.$$

Hence, it takes $O(\rho \log \left\lceil \frac{\alpha k}{\rho} \right\rceil + \rho)$ time to determine the new slot areas for all components when generating the α configurations. Note that $\rho \leq \delta$. We conclude this section with the following result.

Theorem 5.1 *Given the polygon graph G_p, the component graph G_c, the level-linked finger search tree of the slot areas, and the reachability lists, a minimum left-compressed configuration of a given feasible configuration can be generated in $O(\delta(\log n + \log \lceil \frac{\alpha k}{\rho} \rceil))$ time.*

6 Conclusions

In this paper we presented a characterization of a left-compressed, minimum configuration that was the basis for two forbidden region algorithms. The framework underlying these algorithms can be used to solve other, related compaction problems. Consider the problem in which every polygon has its own set of forbidden regions associated. Our algorithms can be used to solve this type of problem within the same time bound. All of our algorithms dealt with rectilinear objects. However, no step of our algorithms depends crucially on this fact. Algorithms based on a similar approach can handle compaction of object of other shapes (e.g., layout components containing 45^o angles) as intersections can be detected efficiently.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] D.G. Boyer. Symbolic layout compaction review. In *Proceedings of 25th ACM/IEEE Design Automation Conference*, pages 383–389, 1988.

[3] Y.E. Cho. Subjective review of compaction. In *Proceedings of 22th ACM/IEEE Design Automation Conference*, pages 396–404, 1985.

[4] G.N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16:1004–1022, 1987.

[5] G.T. Hamachi and J.K Ousterhout. Magic's obstacle-avoiding global router. In *Proc. of 1985 Chapel Hill VLSI Conference*, pages 145–164, 1985.

[6] S. E. Hambrusch and H.-Y. Tu. 1-d compaction in the presence of forbidden regions. In *Proceedings of 28-th Allerton Conference on Communication, Control, and Computing*, pages 828–837, October 1990.

[7] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley & Sons, 1990.

[8] K. Mehlhorn. *Data Structures and Algorithm 1: Sorting and Searching.* Springer-Verlag, 1984.

[9] D. A. Mlynski and C. H. Sung. Layout compaction. In T. Ohtsuki, editor, *Layout Design and Verification*, pages 199–235. Elsevier Science Publ., 1986.

[10] F.P. Preparata and M.I. Shamos. *Computational Geometry.* Springer Verlag, 1985.